Analog Devices公司DSP器件和开发工具的使用技术指南
联系中国DSP免费技术支持中心：(800) ANALOG-D或发邮件：dsp.support@analog.com
查看我们的在线资源：http://www.analog.com/dsp and http://www.analog.com/dsp/EZAnswers

ADSP–TS101S TigerSHARC®处理器启动载入程序核心的操作

Boris Lerner供稿 *2003年4月1日*

简介

本工程师对话介绍了ADSP-TS101S TigerSHARC®处理器的加电启动过程和启动载入程序内核的功能操作。载入程序内核是一个由DSP执行的程序，由VisualDSP++™开发工具的链接实用程序(linker.exe)附在应用代码的前面，实用程序在DSP启动时执行，使处理器可以初始化在应用代码中定义的内部和外部存储器部分。

载入程序内核是一个被传送到DSP的内部存储器的自修改程序。ADSP-TS101支持的三种启动方法是：EPROM启动（通过外部端口）、主机启动（通过外部主机处理器或其它的TigerSHARC）、或链接启动（通过DSP的链接端口）。因此，有三种不同的载入程序内核，支持每种处理器启动模式。

ADSP-TS101S的启动过程

启动模式由DSP的/BMS管脚选择。当处理器处于复位状态时，/BMS管脚是有效输入。如果复位过程中，/BMS的取样是低电平，会选择EPROM模式；在DSP的/RESET信号结束后，/BMS管脚作为EPROM 芯片选择的输出。如果复位过程中，/BMS 的取样是高电平，TigerSHARC将处于IDLE状态，等待主机启动或链接端口启动的发生。

另外，在/BMS管脚上，有一个微弱的内部下拉电阻，不过，请注意，根据这个管脚的外部线荷载，这个下拉可能是不够的。因此，选择EPROM启动模式时，可能需要外部下拉电阻。如果需要主机或链接启动，在复位过程中必须使/BMS保持高电平，并可能需要直接与VCC捆绑。

下面的部分将详细介绍每种启动模式。

EPROM 启动

当选择EPROM启动模式时，TigerSHARC初始化其外部端口DMA通道0，将来自启动EPROM中的256 个32位字的代码送入TigerSHARC的内部存储器块0，位于0x00-0xFF。相应的中断矢量（用于DMA 通道 0）被初始化为0。因此，在DMA完成时，TigerSHARC从位置0x00继续程序执行。这些256字的代码作为启动载入程序，来初始化TigerSHARC存储器的剩余部分。Analog Devices提供带有VisualDSP++开发工具的缺省的启动载入程序内核源文件，称为“TS101_prom.asm”，可以用作参考。

缺省的EPROM启动载入程序和随VisualDSP++工具提供的载入程序实用程序(elfloader.exe)一起工作。载入程序实用程序从用户的工程中将可执行文件(*.*dxe*)和启动载入程序可执行文件(缺省文件是: *TS101_prom.dxe*)中取出，产生EPROM载入程序输出文件(*.*ldr*)。这个载入程序输出文件规定，在启动过程中TigerSHARC的内部和外部存储器的不同块如何被初始化。它的格式如图1所示。块特征字的格式如图2所示。

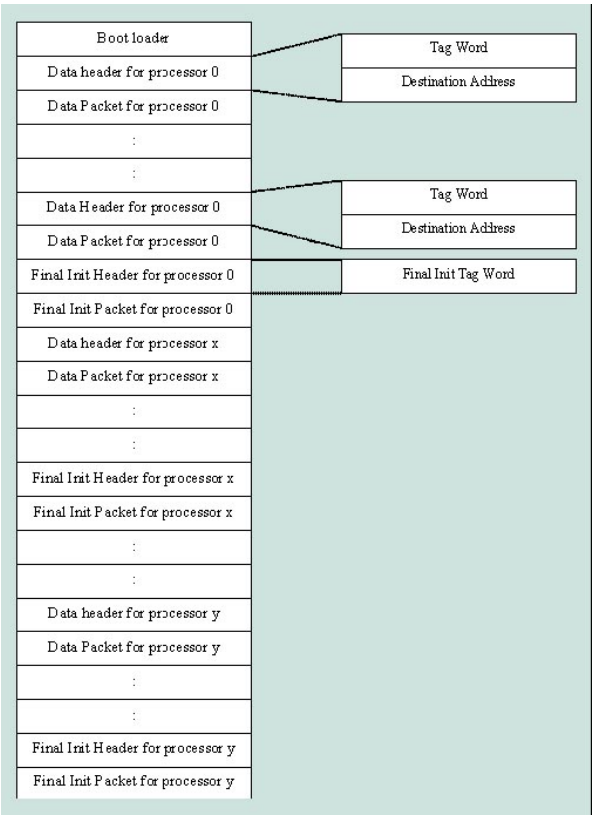
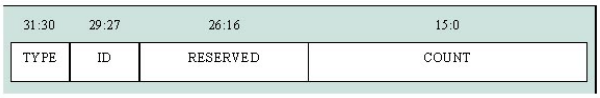


图1 EPROM载入程序格式



Type: 0=Final init, 1=Non-zero init, 2=Zero Init
ID: ID of the processor to which the block belongs
COUNT: Number of 32-bit words in the block

图2 特征字格式

提供的启动载入程序(*TS101_prom.dxe*)按如下描述工作：

1. 启动载入程序内核加载到DSP的内部存储器的块0后，DMA0中断使TigerSHARC从闲置状态退出。然后，DSP开始执行位于0x00000000的启动载入程序。此时，TigerSHARC处于DMA0的中断级，因此，进一步的DMA0 和全局(PMASK[60])中断被禁止。
2. 启动载入程序内核设置SQCTL 寄存器中的NMOD位，确保DSP处于监视模式。RDS指令将目前的中断降低为子程序级。接下来，DMA0和全局中断又被激活。
3. 配置DMA0，从启动EPROM (0x0000-0x03ff是启动载入程序)的地址0x0400开始，将数据移到开始地址为0x00000000的DSP的内部存储器。DMA程序通过对TCB编程，开始DMA，使prom指针提前，处于IDLE状态，直到DMA中断唤醒它，并发送程序序列器指向DMA0中断矢量。此时，RTI指令返回DMA程序，相应地，返回继续执行启动载入程序内核。
4. 由于这不是一个链接端口启动，所有的链接端口DMA通道控制寄存器被复位，所有的链接端口DMA被禁止。
5. 计算处理器ID (这个处理器的)，被计算出并存储在寄存器xR10中。
6. 下一步，启动载入程序内核对来自EPROM的数据块进行语法分析。两个字被移到存储器位置0x00000000和0x00000001。它们是随后的块特征字。在第一个字中，位31:30是块TYPE (0=最后初始化，1=非零初始化，2=零初始化)，位29:27 是处理器 ID，位26:16是保留位，位15:0是块COUNT。第二个特征字是指向DESTINATION地址的指针。
7. 块的ID与存储在寄存器xR10中的处理器ID进行比较。如果两个ID不相同，那么跳过EPROM块。使用来自TYPE和COUNT的值，获得需要跳过的块的尺寸。
8. 如果ID相同，那么检查TYPE.
9. 如果TYPE是1，一次移一个字，将字的

COUNT 数从位置0x00000000，移到DESTINATION地址。完成后，算法返回第5步，并重复该步骤。

10. 如果TYPE是2，将COUNT个零数值移到DESTINATION地址。完成后，算法返回第5步，并重复该步骤。

11. 如果TYPE是0，启动载入程序内核进行“最后的初始化”，即，使用用户应用程序覆盖启动程序的内容。从IDLE状态下唤醒的、最后送入地址0x00000000-0x000000ff的256字的DMA将进行这个操作。不过，将从DMA0中断级开始执行用户应用代码。为避免这种情况，使用下面的算法：

    a. 用户应用代码的前四个指令（预定位于位置0x00000000-0x00000003）从EPROM DMA，存储在寄存器xR11:8中。

    b. 下面的代码被写入位置0x00000000-0x00000003：

```
RETI=0;;
NOP;;
RTI(NP);
Q[j31+=0]=xR11:8;;
```

    c. DMA0中断矢量被设置为0x00000000.

    d. 分支目标缓冲器(BTBINV)不起作用，以清除任何高速缓冲存储分支。

    e. 设置DMA来传送252字的用户代码，预定位置是0x00000004-0x000000ff。

    f. 通过向TCB中写入来开始DMA，然后处理器处于IDLE状态。

    g. 完成DMA后，DMA完成中断将TigerSHARC唤醒，程序序列器跳到b步骤中插入到0x00000000的代码，开始该代码的执行。

    h. 这个代码执行下面的指令：

```
RTI(NP);
Q[j31+=0]=xR11:8;;
```

没有中断级，将用户应用代码放入位置0x00000000-0x00000003，一个序列器跳到地址0x00000000，继续执行（由于RETI被置为0x00000000）。用户应用代码在位置0x00000000开始，没有中断级。注意需要“no predict” (NP)选项，使得这个RTI指令不高速缓冲入BTB。

请注意，如果外部存储器需要特殊的设置（如SDRAM）、需要被载入程序内核初始化。那么，在启动载入程序内核中这个存储器的配置应该发生在其初始化前，这样，用户就需要根据特定的应用和系统要求，修改并重新组建启动载入程序（*TS101_prom.asm*）。

主机启动

选择主机或链接启动时，复位后，TigerSHARC进入闲置状态，等待外部主机处理器或链接端口启动它。主机启动使用TigerSHARC的AUTODMA(任一个通道)，每个AUTODMA都被初始化，以便向TigerSHARC的内部存储器块0传送256字的代码，位置是0x00-0xFF。相应的中断矢量被初始化，指向地址0x00。因此，在DMA完成后，TigerSHARC在存储器位置0x00继续执行。这些256字的代码作为启动载入程序，来初始化TigerSHARC存储器的剩余部分。Analog Devices随VisualDSP++开发工具，提供缺省的启动载入程序，称为“TS101_host.asm”。

缺省的主机启动载入程序和随VisualDSP++工具提供的载入程序实用程序一起工作。载入程序实用程序从用户的工程中将可执行文件和启动载入程序可执行文件(“TS101_host.dxe”)取出，产生主机载入程序文件(文件扩展名为*.LDR)。这个主机载入程序文件说明TigerSHARC的内部和外部存储器的不同块如何被初始化。它的格

式如图1所示。块特征字的格式如图2所示。

在下面的步骤中，AUTODMA0寄存器可以被AUTODMA1寄存器替代，对启动载入程序代码进行合适的修改，以便在寄存器的使用中反映这个变化。

提供的启动载入程序使用AUTODMA0寄存器，其操作如下所述：

1. 加载启动载入程序后，AUTODMA0中断唤醒TigerSHARC，开始在位置0x00000000执行主机载入程序内核。此时，TigerSHARC处于AUTODMA0的中断级，因此，这个通道内的进一步AUTODMA0和全局(PMASK[60])中断被禁止。

2. 复位时，ADSP-TS101 AUTODMA0通道被初始化，以进行四倍字长DMA。这个启动载入程序使用单倍字长DMA，以便进行*.LDR文件的块语法分析（可能不是四倍字长对准的）。因此，任何加载前，主机必须正确初始化AUTODMA0 TCB寄存器。这个启动载入程序位于ASCII 或INCLUDE文件中，占据位置0x0000 - 0x0ff。因此，主机必须首先初始化AUTODMA0 TCB，将256字的数据按普通字（即32位）格式移到TigerSHARC的0x00000000-0x000000ff处。然后，产生一个指向地址0x00000000（缺省时，AUTODMA0中断矢量已经预先指向地址0x00000000）的中断。由于复位时TCB被激活，而且向激活的TCB写入会引起错误，首先必须禁止TCB。因此，下面的值必须加载到TCB中：

   要禁止TCB：

   ```
   DP=0x00000000；；
   DX,DY,DI-do not matter
   ```

   要重新激活TCB：

   ```
   DP=0x53000000
   DY-does not matter
   DX=0x01000001
   DI=0x00000000
   ```

3. 主机使用刚刚初始化的AUTODMA0通道，将*. LDR文件的所有字传送到TigerSHARC。这个载入程序文件的第一个256字被AUTODMA到TigerSHARC的存储器位置0x00000000-0x000000ff处。此时，中断接过控制权，使程序序列器在位置0x00000000开始执行，即载入程序的开始处。此时，TigerSHARC处于AUTODMA0的中断级，因此，进一步的AUTODMA0和全局(PMASK[60])中断被禁止。主机随后发送的字被载入程序进行语法分析，如下所述。

4. 载入程序在SQCTL寄存器中设置NMOD位，确保处于监视模式。然后，RDS指令将中断降低到子程序级。接下来，AUTODMA0和全局中断被重新激活。

5. AUTODMA0中断矢量被设成标签“_dma_int”的地址。

6. 计算处理器ID，并存储在寄存器xR10中。

7. 由于这不是一个链接端口启动，所有链接端口控制都被复位，所有链接端口的DMA被禁止。

8. 寄存器xR3:0将被用于开始AUTODMA0。寄存器xR1包含会改变和修改的计数，通常是1，因此在开始AUTODMA0之前，将单独设置xR1。寄存器xR3被置为0x53000000，xR0被置为0x00000000，即正常字，中断激活，TigerSHARC内部存储器的目标地址0x00000000。

9. 下一步，载入程序对来自主机的数据块进行语法分析。它的设置是发送两个字。这些是随后的块的特征字。在第一个字中，位31:30是块TYPE (0=最后初始化，1=非零初始化，2=零初始化)，位29:27是处理器

ID，位26:16是保留位，位15:0是块COUNT。第二个特征字是指向DESTINATION地址的指针。

10. 块的ID与存储在xR10中的ID比较。如果ID不同，块被跳过（如果TYPE=0或1，不存储，从主机获得255或字的COUNT值），从步骤9重复进行。
11. 如果ID相同，检查TYPE字段。
12. 如果TYPE字段是1，通过AUTODMA0，一次一个字，将字的COUNT值移到DESTINATION。完成后，从步骤9重复进行。
13. 如果TYPE字段是2，将COUNT个零数值移到DESTINATION地址。完成后，从步骤9重复进行。
14. 如果TYPE字段是0，载入程序进行“最后的初始化”，即，使用用户应用代码覆盖启动程序的内容。从IDLE状态下唤醒的、最后送入地址0x00000000-0x000000ff的256字的AUTODMA0将进行这个操作。不过，将从中断级AUTODMA0，开始执行用户应用代码。为避免这种情况，使用下面的算法：
    a. 用户代码的前四个指令（位于位置0x00000000-0x00000003）从主机AUTODMA，并存储在寄存器xR11:8中。
    b. 下面的代码被写入位置0x00000000-0x00000003：

```
RETI=0;;
IMASKH=yR0;;
RTI (NP);
Q[j31+=0]=xR11:8;;
```

    c. AUTODMA0中断矢量被设置为0x00000000.
    d. 寄存器yR0被预设成值0x80000000。这将是IMASKH的新值，以禁止除了仿真之外的所有中断。
    e. 使分支目标缓冲器(BTBINV)不能清除任何高速缓冲存储分支。
    f. 设置AUTODMA0来传送252字的用户代码，位置是0x00000004-0x000000ff。
    g. 通过向TCB中写入来开始AUTODMA0，然后处理器进入IDLE状态。
    h. 完成AUTODMA0后，其中断将TigerSHARC唤醒，跳到插入到b步骤中的0x00000000的代码，开始执行。
    i. 执行下面的指令：

```
IMASKH=yR0;;
```

禁止所有的全局中断，然后执行下面的命令行：

```
RTI(NP);
Q[j31+=0]=xR11:8;;
```

将中断级置为没有，将用户代码放入位置0x00000000-0x00000003，将执行跳到0x00000000（由于RETI被置为 0x00000000）。用户代码在位置0x00000000开始，没有中断级。注意需要(NP)选项，使得RTI不高速缓冲入BTB。

请注意，如果需要特殊设置（如SDRAM）的外部存储器、需要被载入程序初始化，那么，在启动载入程序中，这个存储器的设置需要发生在其初始化之前，这样，用户就需要修改并重新组建启动载入程序。

另外，注意主机不要使AUTODMA0缓冲区过度使用。因此，主机需要监视AUTODMA0的状态，仅当状态是“激活”时（即DMA已经释放了AUTODMA0缓冲器），才开始一个新的DMA。在这个监视过程中，注意在允许对外部总线进行DSP访问时，主机应该当心。否则，

将发生软件死锁，（即DSP永远不会完成一个外部事务，主机永远不会得到开始新事务的通知）。

在VisualDSP实例中，提供了一个TigerSHARC作为主控者、主机启动另一个TigerSHARC的实例代码；源代码在“mpmaster.asm”文件中。

链接启动

选择主机或链接启动时，在复位后，TigerSHARC进入闲置状态，等待主机或链接端口启动它。链接启动可以使用TigerSHARC的任何链接端口，每个链接端口的DMA都被初始化，以便向TigerSHARC的存储器块0传送256字的代码，位置是0x00-0xFF。相应的DMA中断矢量被初始化为0。因此，在链接DMA完成后，TigerSHARC在存储器位置0x00继续执行。这些256字的代码作为启动载入程序，来初始化TigerSHARC存储器的剩余部分。Analog Devices随VisualDSP开发工具，提供缺省的启动载入程序，称为“TS101_link.asm”。

缺省的链接启动载入程序和随VisualDSP提供的载入程序实用程序一起工作。载入程序实用程序从用户的工程中将可执行文件(*.DXE)和启动载入程序可执行文件（缺省是：“TS101_host.dxe”)取出，产生链接载入程序文件输出文件(*.LDR)。这个LDR文件规定TigerSHARC的内部和外部存储器的不同块如何被初始化。它的格式如图1所示。块特征字的格式如图2所示。

提供的启动载入程序的工作如下：

1. 加载启动载入程序之后，链接端口的DMA中断唤醒TigerSHARC，开始执行位于0x00000000的载入程序。此时，TigerSHARC处于链接端口DMA的中断级，因此，进一步的链接端口DMA和全局(PMASK[60])中断被禁止。
2. 常数LINK定义使用那个链接端口进行启动。在这个代码中，它被置为1（即链接端口1）。如果使用不同的链接端口，必须改变常数的值，重新组建代码。
3. 载入程序在SQCTL寄存器中设置NMOD位，确保处于监视模式。然后，RDS指令将中断降低到子程序级。接下来，链接端口DMA和全局中断(PMASK[60])被重新激活。
4. 链接端口接收DMA中断矢量被设置成标签“_dma_int”的地址。不使用的链接端口被清除和禁止，链接端口DMA被禁止。
5. 链接端口控制寄存器被初始化。
6. 将数据从链接端口送入的DMA，将进行一次一个四位字长的操作。寄存器XR3:0被预先设置 到TCB需要的值。
7. 来自链接端口的数据被子程序“_read_word”读取。来自链接端口的数据总是四倍字长格式，但处理器需要一次对一个32位字进行语法分析，需要保持一个内部FIFO缓冲器。这是在存储器的位置0x00-0x03，作为循环缓冲器实现的，寄存器J2作为指向缓冲器的读指针，因此，依次对J2、JB2和JL2初始化。“_read_word”的执行流程如下：
   a. 首先检查J2看其是否恢复到零（即缓冲器中 所有的数据都被读取），如果否，转到步骤“d”，从缓冲器读取下一条数据。
   b. 另一个四位字从链接端口送入缓冲器。为避免数据刚好在DMA开始时送入（由于硅芯片错误#147，这会引起数据丢失），在循环中监视相应的LSTAT，等待接收缓冲器添满。然后，通过将寄存器XR3:0写入TCB，开始相应的链接端口DMA，程序在IDLE

中等待DMA中断。

c. 当新四位字从链接端口到来时，DMA中断将处理器从IDLE唤醒，执行转到分支“_dma_int”，“NOP;;”指令后跟着一个“RTI;;”指令，将它送回到IDLE后的一个指令。（*注意，这里“NOP;;”指令是必需的，因为不允许“RTI;;”指令是ISR的第一个指令*）。

d. 由J2指向的来自缓冲器的数据，被读入xR4，J2循环增加。

8. 与其它启动模式不同，因为这个载入程序不支持多处理器(MP)启动，这里不使用处理器ID。

9. 下一步，载入程序对来自链接端口的数据块进行语法分析。两个字被移到yR8和J0。这些是随后的块的特征字。在第一个字中，位31:30是块TYPE (0=最后初始化, 1=非零初始化, 2=零初始化)，位29:16是保留位，位15:0是块COUNT。第二个特征字是指向DESTINATION地址的指针。

10. 如果TYPE 是1, 通过“_read_word”，一次一个字将字的COUNT数从位置0x00000000移到DESTINATION地址。完成后，算法转到第9步。

11. 如果TYPE是2，零的COUNT数被移到DESTINATION地址。完成后，算法转到第9步。

12. 如果TYPE是0，启动载入程序进行“最后的初始化”，即，使用用户应用代码重写自身的内容。使用下面的算法：

a. 通过“_read_word”，将用户应用代码的前28个指令（预定目标位置是0x00000000-0x0000001B）从链接端口移出，并存储到寄存器xR31:8和yR31:28。

b. 位于“_dma_int”的中断服务程序被重新定位到0x04-0x05。

c. 子程序“_last_patch_code”的22个指令被重新定位到位置0x06-0x1B。

d. 使分支目标缓冲器(BTBINV)不能清除高速缓冲存储分支。

e. 链接端口中断矢量被置为地址0x04（作为步骤b的结果，现在这个位置包括中断服务程序）。

f. 寄存器yR1被初始化为0x80000000；这个值将写入IMASKH，以便在用户代码的开始禁止全局中断（注意仿真中断是激活的）。

g. 寄存器J0被初始化到0x1C (重新定位的“_last_patch_code”后的第一个位置)，LC0被初始化到 0x E4 (最后初始化中需要读取的字的数目)。

h. 此时，位置0x04-0x1B的初始化如下进行：

```
0x04: _relocated_dma_int:
nop;;
rti(NP);;

0x06: _relocated_read_word:
// if J2 -> start of the buffer...
comp(j2,0);;
// ...bring in more data
if njeq, jump _relocated_read_buffer (NP);;

_relocated_wait_for_data:
yr2 = LSTATx;;
ybitest r2 by 3;;
if ySEQ, jump _relocated_wait_for_data (NP);;

// start the DMA
DCx = xr3:0;;
// wait till DMA interrupts
```

```
idle;;

 _relocated_read_buffer:
// read the word from the buffer
xr4 = cb[j2+=1];;
// and return
cjmp (ABS) (NP);;

0x0F: _relocated_final_init1:
//read word
call _read_word (NP);;

// write it
[j0 += 1] = xr4;;
if NLC0E, jump _relocated_final_init1 (NP);;

 // disable all ints except emulation
IMASKH = yr1;;
// Link disable and clear
LCTLx = yr0;;

 // overwrite 0x00-0x03
Q[j31 + 0] = xr11:8;;
// overwrite 0x04-0x07
Q[j31 + 4] = xr15:12;;
// overwrite 0x08-0x0b
Q[j31 + 8] = xr19:16;;
// overwrite 0x0c-0x0f
Q[j31 + 0xc] = xr23:20;;
// overwrite 0x10-0x130

Q[j31 + 0x10] = xr27:24;;
// overwrite 0x14-0x17
Q[j31 + 0x14] = xr31:28;;

// overwrite 0x18-0x1b, start at 0
jump 0 (ABS) (NP); Q[j31 + 0x18] = yr31:28;;
```

i. 代码的执行跳到0x0F，即上面所示的“_relocated_final_init1”。

j. 位置0x1C-0xFF被来自链接端口的数据填充。注意，“_relocated_final_init1”处的指令“call_read_word (NP);;”，是相对调用指令。因此，它实际上调用“_relocated_read_word”，改写旧代码“_read_word”，不会引起任何问题。

k. 现在，链接端口的接收结束了，正确的数据在0x1C-0xFF中，应该位于0x00-0x1B的数据在寄存器xR31:8和yR31:28中。剩下的代码用xR31:8中的数据改写存储器位置0x00-0x17，最后，当执行到0x00的绝对跳转时，代码的最0x1B (包括其自身)。后一行用来自yR31:28的数据，改写位置0x18-0x1B。

l. 用户代码从0x00开始。

请注意，如果需要特殊设置（如SDRAM）的外部存储器、需要被载入程序初始化，那么，在启动载入程序中该存储器的设置应该发生在其初始化前，这样，用户就需要修改并重新组建启动载入程序（TS101_link.asm）。

文件历史

版本	描述
2003年4月1日，G. Fowler.供稿	增加了简介，并对文档进行了重新组织。
2002年2月1日，B. Lerner.供稿	第一版